Conception d'un systeme de bases de donnees relationnel multiprocesseur

Georges Gardarin, Ph. Bernadat, N. Temmerman, Patrick Valduriez, Y. Viemont

INRIA
CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tél. (3) 954 90 20

Rapports de Recherche

N° 224

CONCEPTION D'UN SYSTÈME DE BASES DE DONNÉES RELATIONNEL MULTIPROCESSEUR

Georges GARDARIN
Philippe BERNADAT
Nicole TEMMERMAN
Patrick VALDURIEZ
Yann VIEMONT

Juillet 1983

CONCEPTION D'UN SYSTEME DE BASES DE DONNEES RELATIONNEL MULTIPROCESSEUR

Georges GARDARIN, Philippe BERNADAT
Nicole TEMMERMAN, Patrick VALDURIEZ, Yann VIEMONT

Université de Paris VI et INRIA
Projet SABRE, BP.105, 78153 LE CHESNAY-Cédex, France

SABRE (Système d'Accès à des Bases RElationnelles) est un système de bases de données relationnelles portable, principalement conçu pour une configuration multiprocesseurs. Ce système, dont une première génération est opérationnelle à l'INRIA sur MULTICS, présente plusieurs idées originales dans les domaines suivants : regroupement de données, vues multiples des bases de données, algorithmes pour les jointures de relations, contrôles de concurrence, intégrité, optimisation de questions. Cette communication résume les objectifs, les concepts de base et les architectures fonctionnelles et opérationnelles de SABRE.

DESIGN OF A MULTIPROCESSOR RELATIONAL DATABASE SYSTEM

Georges GARDARIN, Philippe BERNADAT
Nicole TEMMERMAN, Patrick VALDURIEZ, Yann VIEMONT

Université de Paris VI et INRIA
Projet SABRE, BP.105, 78153 LE CHESNAY-Cédex, France

SABRE (Système d'Accès à des Bases RElationnelles) is a portable data management system, mainly designed for a multi-microprocessor configuration. This system, of which a first generation is actually operational at INRIA on MULTICS, presents several original ideas in the following areas : clustering of data, multiple views of a data base, algorithms for relational joins, concurrency control, integrity control, and query optimization. This paper summarizes the objectives, and the functional and operational architectures of the SABRE system.

Articles présentés à IFIP'83,
Paris, Septembre 1983.

DESIGN OF A MULTIPROCESSOR RELATIONAL DATABASE SYSTEM

Georges GARDARIN, Philippe BERNADAT
Nicole TEMMERMAN, Patrick VALDURIEZ, Yann VIEMONT

University of Paris VI and INRIA
SABRE Project, BP.105, 78153 LE CHESNAY-Cédex, France

SABRE (Système d'Accès à des Bases RElationnelles) is a portable data management system, mainly designed for a multi-microprocessor configuration. This system, of which a first generation is actually operational at INRIA on MULTICS, presents several original ideas in the following areas : clustering of data, multiple views of a data base, algorithms for relational joins, concurrency control, integrity control, and query optimization. This paper summarizes the objectives, and the functional and operational architectures of the SABRE system.

1. INTRODUCTION

A new class of data base management systems characterized by high level manipulation languages and based on the relational model is actually being commercialized {1,2}. More functionality is provided to the user but with three main drawbacks : high response times, impossibility of integrating such systems into a computer network, and management restricted to alphanumerical formatted data.

To avoid the first drawback of conventional systems, proposals have been made to relieve the main computer of the data processing functions to be done by a so-called database machine. We distinguish such database machines as being either hardware-oriented systems or software-oriented systems. Hardware-oriented systems use special purpose hardware to perform all the database management functions. CAFS {3} and VERSO {4} are examples of such systems. Software-oriented systems are those wherein most of the database management functions are done in software, although a small amount of special-purpose hardware may be used. DIRECT{5} and MBDS {6} are examples of the latter. The rapid growth in the development of low-cost and powerful general-purpose processors suggests that this second approach is much more promising.

The SABRE system {7} described herein is a software-oriented system. Its first aim is to avoid the drawbacks of high response times and difficulties of integrating such a system into a computer network. To improve the performance of the database management functions, we investigate several approaches : utilization of parallelism, filtering of data on the fly, new access path methods allowing efficient clustering of data, and large memory caching disks. To avoid the second drawback, we develop SABRE as a system portable on different computers : big computers as HB-68 MULTICS, as well as autonomous multi-microprocessor machines. Therefore, the system is written in PASCAL, making it easily portable.

The project aims to develop an autonomous system implemented on a multi-microprocessor machine and connectable to a local or general network. This system will be a basic tool for investigating on the management of integrated databases, where different data types have to be handled.

2. OBJECTIVES OF SABRE

The SABRE project is characterized by the following objectives :

(1) To develop an extensible and portable French relational data base manager ; this system should be a basic tool to build integrated database systems and/or distributed database systems.

(2) To improve response time in comparison with classical relational data base systems. Thus, we investigate :

- (2a) The possibilities of parallelism : a request is divided into a set of functional units which are executed in parallel ;
- (2b) The possibilities of reducing the I/O time. Two devices are utilized : a large cache memory where intermediate and final results of data manipulation are stored, and a filtering processor which performs selections on the fly partly in parallel to data transfer from disk to cache {4}.
- (2c) The management of efficient access paths to relations ; we believe that B-trees are not well adapted for multi-attribute queries and we propose a new structure called predicate trees to accelerate the access to data.

(3) To allow various views of the data base to be defined and queried by different groups of users ; each view can contain virtual relations derived from other views but also actual data known only at the view level.

(4) To guarantee the database integrity when data are updated simultaneously by concurrent transactions or when erroneous updates occur.

(5) To obtain a slow degradation of the system performance without lost of functionality when a component of the system fails.

More specifically, we try to fulfill these seven objectives by developping a software system and by using off-the-shelf hardware. Thus,

certain objectives will only be partly accomplished by the end of the project, but we hope to give coherent partial solutions to all of them.

3. FUNCTIONAL ARCHITECTURE OF SABRE

3.1. Basic principles

We distinguish the functional architecture of SABRE which is unique, from the operational architectures of the various implemented versions of SABRE. (There is one operational architecture for each version of SABRE.) The functional architecture is composed of a set of successive layers from the end user to the disk units. Each layer is characterized by a set of operations on objects which constitutes its external interface. The interface of a layer i can be called by all external layers. Going from the end user to the disks, each layer manipulates simpler and simpler objects. An operational architecture is composed of a set of processors which may be organized in classes of parallel processors and which are generally interconnected by buses. A class of processors corresponds to a layer of the functional architecture. The layers of the functional architecture are presented in a top down fashion. The execution of a typical query through the layers is depicted in fig. 1.

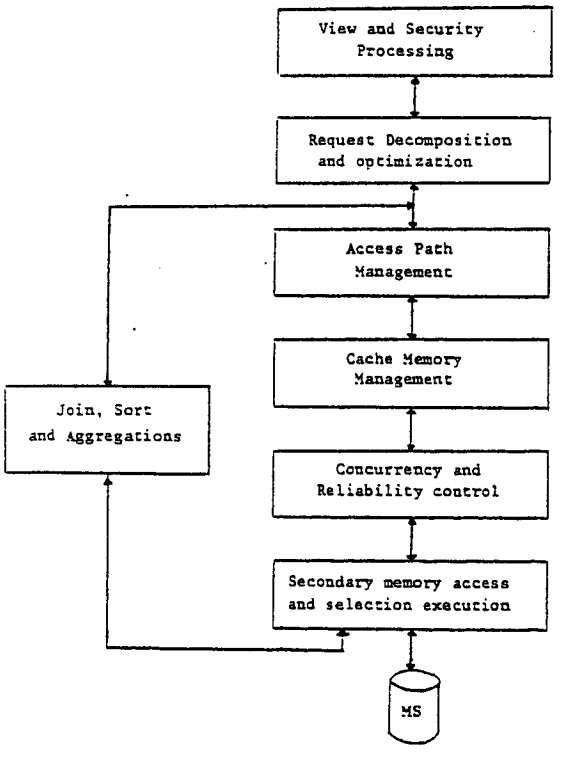


Fig. 1 : A typical query execution

3.2. End user interface layer

The end user interface is supplied by the first layer. Its role is to enter the queries or updates on a database view, then to parse the requests and to send them to the next layer. It also performs the presentation of the results to the users. The external language is a parametrized flexible language which provides the user with both SQL like or QUEL like queries. The parsing of the requests is based on an augmented transition network {8}. The rest of the machine is designed to be connected to host processors. Thus the first layer should be run by a host processor in a multiprocessor operational architecture. The set of commands and corresponding answers composes an application protocol of the ISO architecture {9}, called a data manipulation protocol. It is composed of transaction control commands and formatted data manipulation commands. In the future, we also wish to extend the protocol with unformatted data manipulations (such as long text or bit strings).

3.3. The view layer

The next layer of the functional architecture manages the views. It mainly maps requests on views into requests on relations of one (or several) underlying bases. The algorithm employed is based on query modification {10}. A view is a set of relations possibly derived from existing relations. A view is associated with a group of end users having rights on certain relations in the view. This layer also performs authorization checking and integrity controls. The external interface of this layer is the data manipulation protocol.

3.4. The base layer

The next layer receives requests on base relations and transforms them into a tree of extended relational algebra operations. It performs the optimization of this tree. We use a heuristic based on a first move of selections at the bottom of the tree. Then, several trees, generated by permutation of join and other operations, are evaluated by considering the I/O time. This evaluation is based on a statistical model permitting the system to estimate the size of the result of any operation. The tree leading to the minimum I/O time is selected. The levels of possible parallelism in the tree are also isolated. The relational algebra operations of the choosen tree are requested by the subsequent layer.

3.5. The relation layer

The next layer performs relational algebra operations on relations. Its first role is to manage access paths to relations in order to determine the parts of one or two relations which must be accessed to perform unary or binary operations. Access paths are maintained with a new data structure called predicate trees {11}, especially designed for multi-key searching. The other task of this layer is to perform some specific functions on parts of relations, mainly the join and sort operations.

3.6. The partition layer

Tuples are stored inside associative partitions. The size of such a partition is generally a disk track, but it can be less. The addressing inside a partition is done by content, using Boolean expressions. A Boolean expression is composed of elementary predicates involving alpha-numerical or textual data.

The partition layer performs the selections on a list of partitions identified by a logical address. Also it performs insertions of new tuples in a partition and deletions of selected tuples. Moreover, it applies concurrency control and commitment of updates at the end of a transaction.

4. POSSIBILITIES OF PARALLELISM

4.1. Vertical parallelism

Parallelism is introduced in the query execution in a vertical manner by assigning a different processor to each layer. In general, different queries will be run in parallel by different processors leading to inter-request parallelism. It is also possible to perform intra-request parallelism by pipelining data transfers. For example, a join operation can start as soon as a first data page of each operand relation is available in core memory. If, in the relational tree, the join operation follows selection operations, then selection and join can be done partly in parallel. In conclusion, vertical parallelism allows the multiprocessor machine to perform both intra and inter-request parallelism.

4.2. Horizontal parallelism

Parallelism is also introduced in a horizontal manner by assigning several processors to each layer. Horizontal parallelism is classicaly done in two ways. Functional parallelism consists in subdividing the functions of a layer in sub-functions and running them in aparallel. For example, the functions of layer 4, which manages views, integrity and authorization can be divided in two sub-classes of sub-functions : query modification and authorization control. These two sub-functions can be executed by two different processors.

Division parallelism consists in subdividing the data to be processed in parts and to give each part to a processor. For example, this kind of parallelism can be applied in performing selections : we can assign a processor, per disk unit or, to some extent, a processor per track to carry out selections. Such a specialized processor is called a filter {4,12}. A filter operates on a subset of a relation. This kind of parallelism is the basis of such machines architectures as DIRECT {5} and CASSM {13}. A more complex application of division parallelism is to perform joins in parallel {14}. In the latter case, one relation is generally divided in parts while the other one can be broadcasted to all join processors.

4.3. Costs and advantages of parallelism

Let us consider a module M which can be divided by one of the previous techniques in two similar or different modules m1 and m2. We can assume that running M on a unique processor requires a time :

$$T(M) = T(m1) + T(m2)$$

Running m1 and m2 on two different processors requires a time which is the minimum of T(m1) and T(m2), plus a synchronization and communication time C(m1,m2). If T(m1) is greater than T(m2), then running m1 and m2 in parallel gives:

$$T(M) = T(m1) + C(m1,m2)$$

This simple estimation shows the value of performing parallelism when $C(m1,m2) < T(m2)$, that is when the execution time of the shortest module is less than the time lost for communication and synchronization. This suggests that parallelism should be introduced as often as possible to isolate a specific function having a high execution time in comparison with the communication and synchronization times. This value of parallelism is the basic justification of the ideal operational architecture of SABRE, which is presented in the next section.

5. THE OPERATIONAL ARCHITECTURES OF SABRE

5.1. The ideal operational architecture

The ideal operational architecture is composed of a set of classes of specialized processors. These processors are called virtual since in a real operational architecture some of them can be implemented on the same real processor. Each processor type is included in a layer of the functional architecture. We allow several occurrences of each processor type to be present when it is necessary to perform division parallelism. The number of specified processors per type depends on the size of the database and the number of users. Moreover, the number of processors could differ for each type of query. The ideal operational architecture is represented in fig. 2.

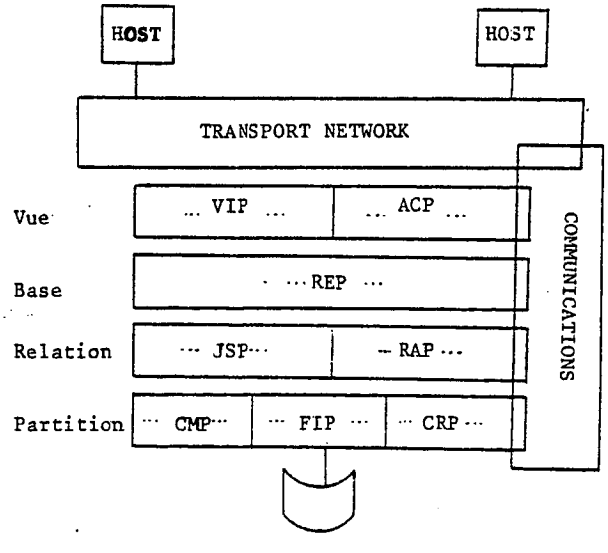


Fig. 2 : The ideal operational architecture of SABRE

The SABRE machine is composed of the following processors :

- View and Integrity Processor (VIP) : This processor is the most external of the machine and works on a database view. Its role is first to translate the request expressed on virtual relations described in the view in a request expressed on real relations implemented on disks. The second role is to perform some

integrity control at each update.

- Authorization Control Processor (ACP) : This processor controls the user rights on the views. For this purpose, it accesses a meta base and checks the presence of correct operation rights on the requested objects.

- Request Evaluation Processor (REP) : This processor works on a set of implemented relations (a real data base) and performs the request decomposition and optimization.

- Relation Access Processor (RAP) : This processor works on an implemented relation and manages access paths. More precisely, when an insertion of tuples is performed, it determines the partitions where to insert the tuples, and manages the predicate trees associated with relations {11}. When a restriction is performed, it determines which partitions should be scanned.

- Join, Sort and Aggregate Processors (JSP) : These processors perform join, sort and compute aggregate functions on partitions. Several join processors generally work in cooperation to efficiently carry out join and sorting {14,15}.

- Concurrency Control and Recovery Processor (CRP) : This processor performs concurrency control using an algorithm based on two time-stamps ; this algorithm ensures a unique update order which is the transaction commit order {16}. It also performs update logs and manages the two-step commit protocol {16}.

- Cache Memory Processor (CMP) : This processor is responsible for allocating secondary and cache memories. It also performs the replacement of cache memory pages into partitions when the cache memory is saturated.

- Filtering Processor (FIP) : These processors perform selection, insertion and deletion of tuples in a partition. They work as much as possible on the fly, during the transfer of data from disk to cache.

5.2. The operational architecture of SABRE Generation 1

A real operational architecture is associated with each implemented version of SABRE. It is composed of one or more real processors supporting a set of virtual processors of the ideal operational architecture. Two generations of SABRE, namely G1 and G2, are currently under implementation. The operational architecture of SABRE.G1 is portrayed fig. 3. All the virtual processors are implemented on one real processor. This multi-user, multi-process and mono-processor generation of SABRE is currently working on MULTICS at INRIA. All the virtual processors are written in PASCAL, and are thus easily transported to another system.

5.3. SABRE Generation 2, a multiprocessor database computer

The operational architecture of SABRE.G2 is presented fig. 4. This is a true multi-microprocessor database computer. It will include three parallel processors and be usable by two users simultaneously in a first version. The software will be the same as the SABRE Generation 1 software. The implementation of this version should start in April 1983.

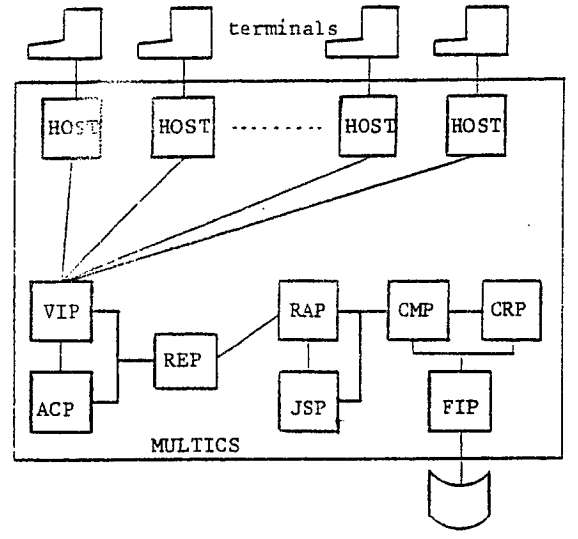


Fig. 3 : Operational architecture of SABRE.G1

The hardware to be used in SABRE.G2 has been developed by the french PTT (CNET) and is called the SM90. It is composed of three 68000 and I/O processor. Each 68000 will run on UNIX and each can address a private memory, a local memory and a common memory through mapping facilities supplied by the MMU. We hope to be able to move virtual processors into the real ones into order to compare their performance.

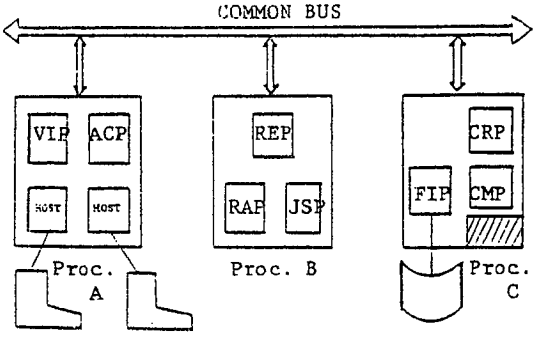


Fig. 4 : Operational architecture of SABRE.G2

6. CONCLUSION

The principal aim of the SABRE project is to improve the performance and functionality of systems that manage large relational databases. The central concepts employed are : utilization of parallelism, filtering of data on the fly, cache memory, multi-dimensional clustering of data, and concurrent commitment of transactions. Another important aim is to build a portable system ; hence the system is written in PASCAL.

The distinction between functional architecture and operational architectures permits functions (required of the system) and hardware (to im-

plement the functions) to be treated independently. Thus, concepts dealing with function optimization have emerged and led to the definition of virtual processors supporting these functions. The set of virtual processors, called the ideal operational architecture, can be mapped into a various number of operational architectures by assigning real processors to one or more virtual processors. Accordingly, the system can be embedded in different computers (HB68, multi-microprocessor ...).

The realization of the system is being accomplished in steps corresponding to successive operational versions of SABRE. This gradual approach began by testing Generation 1, which is a complete relational DBMS based on original ideas. This version has permitted the validation of the interfaces and algorithms in a uniprocessor context. Generation 2 is a relational database system implemented on a multi-microprocessor machine. A future, more sophisticated generation would permit the integration of text and image types of data, thereby providing a new class of database management systems.

6. REFERENCES

{1} D.D. Chamberlin, A.M. Gilbert, R.A. Yost : " A history of System R and SQL/Data System", 7th Int. Conf. on Very Large Data Bases, Cannes, September 1981.

{2} M. Stonebracker, E. Wong, P. Kreps : "The design and implementation of INGRES", ACM Transactions on Data Base Systems, vol. 1, n° 3, September 1976.

{3} E. Babb : "Implementing a relational data base by means of specialized hardware", ACM Transactions on Data Base Systems, vol. 4, n° 1, March 1979, 1-29.

{4} F. Bancilhon, M. Scholl : "Design of a back-end processor for a database machine", Proc. of the ACM-SIGMOD, Santa Monica, California, May 1980.

{5} D.J. Dewitt : "Query execution in direct", Proc. of the ACM-SIGMOD, Int. Conf. of Management of Data, May 1979, 13-22.

{6} D.K. Hsiao, M.J. Menon : "Design and analysis of a multi-backend database system for performance improvement, functionality expansion and capacity growth", Technical Report OSU-CISRC-TR-81-7, Ohio State University, Columbus, July 1981.

{7} G. Gardarin : "An introduction to SABRE : a multi-microprocessor database machine", 6th Workshop on Computer Architecture for Non Numeric Processing, Hyères, France, June 1981.

{8} W.A. Woods : "Transaction network grammar for natural analysis", Comm. ACM, vol. 13, n° 10, October 1970.

{9} ISO/TC97/SC16 N227 : "Reference model of open system interconnection", June 1979.

{10} M. Stonebracker : "Implementation of integrity constraints and views by query modification" : Proc. of the ACM-SIGMOD Workshop on Management of Data, San Jose, California, May 1975.

{11} G. Gardarin, P. Valduriez, Y. Viémont : "Predicate trees", Internal Report, SABRE project, INRIA, March 1983.

{12} P. Faudemay : "Sur une nouvelle classe de filtres multi-expressions", Journées Machines Bases de Données, Sophia-Antipolis, Septembre 1980.

{13} S.Y.W. Su, G.J. Lipovski : "CASSM : a cellular system for large data bases", Proc. Int. Conf. on Very Large Data Bases, September 1975.

{14} P. Valduriez, G. Gardarin : "Multiprocessor join algorithms of relations", 2nd Int. Conf. on Data Bases : Improving Usability and Responsiveness, Jerusalem, June 1982.

{15} P. Valduriez : "Semi-join algorithms for multi-processor systems", ACM-SIGMOD Int. Conf. on Management of Data, Orlando, Florida, June 1982.

{16} Y. Viémont, G. Gardarin : "A distributed concurrency control based on transaction commit ordering", 12th Int. Conf. on FTCS, Los Angeles, June 1982.

CONCEPTION D'UN SYSTEME DE BASES DE DONNEES RELATIONNEL MULTIPROCESSEUR

Georges GARDARIN, Philippe BERNADAT
Nicole TEMMERMAN, Patrick VALDURIEZ, Yann VIEMONT

Université de Paris VI et INRIA
Projet SABRE, BP.105, 78153 LE CHESNAY-Cédex, France

SABRE (Système d'Accès à des Bases RElationnelles) est un système de bases de données relationnelles portable, principalement conçu pour une configuration multiprocesseur. Ce système, dont une première génération est opérationnelle à l'INRIA sur MULTICS, présente plusieurs idées originales dans les domaines suivants : regroupement de données, vues multiples des bases de données, algorithmes pour les jointures de relations, contrôles de concurrence, intégrité, optimisation de questions. Cette communication résume les objectifs, les concepts de base et les architectures fonctionnelles et opérationnelles de SABRE.

1. INTRODUCTION

Plusieurs systèmes de gestion de bases de données caractérisés par des langages de manipulation de haut niveau permettant d'accéder à des ensembles de données, généralement basés sur le modèle relationnel, sont en cours de commercialisation {1,2}. Bien que très active dans le domaine des bases de données réparties {3}, la recherche française fut à peu près absente lors des développements de ces systèmes centralisés nouveaux (1974-1980). D'ores et déjà, les systèmes relationnels actuels présentent trois inconvénients importants : restriction des bases à des données alphanumériques structurées, problèmes de temps de réponse élevés, non possibilité d'intégration des systèmes proposés dans un réseau d'ordinateurs.

Le projet SABRE {4} se propose dans un premier temps de répondre à deux inconvénients importants des systèmes de bases de données relationnels actuels : problèmes de temps de réponse élevés, non possibilité d'intégration des systèmes proposés dans un réseau d'ordinateurs. Afin de répondre à la première limitation, nous explorons divers types d'approches : utilisation du parallélisme, filtrage à la volée des données, nouvelles méthodes d'accès permettant un regroupement physique plus pertinent des données, anté-mémoire de taille importante placée devant les disques ... Afin de répondre au deuxième inconvénient, nous développons le système SABRE comme un système portable sur plusieurs types d'ordinateurs : gros ordinateurs type HB.68 MULTICS, mais aussi multi-microprocesseur autonome. Dans ce but le système est décrit en PASCAL facilement portable.

L'aboutissement du projet, vers un système autonome supporté par un multi-microprocesseur et connectable à un réseau local ou général, devrait permettre d'aborder les systèmes de bases d'informations futurs. Ceux-ci devraient, à notre sens, bien évidemment supporter des accès assertionnels à des bases relationnelles avec de bons temps de réponse (quelques dizaines de requêtes par minute) mais aussi permettre d'intégrer divers types de données non structurées : textes, images, figures géométriques ... Ces systèmes devraient être la base des systèmes de CAO et des systèmes experts du futur.

Cette présentation générale du système SABRE est organisée comme suit. Tout d'abord les objectifs du projet sont rappelés. Ensuite, un effort est fait afin de dégager quelques concepts de base essentiels. Puis l'architecture fonctionnelle des systèmes SABRE est présentée dans la Section 4. La première génération des systèmes SABRE est caractérisée par l'utilisation de systèmes supports mono-processeurs classiques. La deuxième génération devrait exploiter le parallélisme possible avec une machine multiprocesseur. Les architectures opérationnelles de ces deux générations sont introduites et situées par rapport à une architecture opérationnelle idéale dans la Section 5.

2. OBJECTIFS

Les principaux objectifs retenus par le projet SABRE sont les suivants :

(1) Fournir un outil portable et extensible, pierre de base pour construire des systèmes de gestion de bases de données intégrées (données structurées, textuelles, images) et/ou réparties.

(2) Permettre un accès assertionnel intelligent (l'usager doit décrire les données qu'il désire et non la manière de les obtenir) à des vues (parties de la base structurée selon le désir des usagers) d'une base de données relationnelle.

(3) Améliorer les temps de réponse par utilisation du parallélisme intra-requête (une même requête peut être traitée en parallèle par plusieurs processeurs) et inter-requêtes (des requêtes de différentes transactions peuvent être traitées en parallèle). On vise à titre indicatif d'être capable, avec des versions multiprocesseurs, d'atteindre des débits de l'ordre de 10 interrogations complexes par minute.

(4) Diminuer le nombre d'Entrées/Sorties et le temps perdu à effectuer ces Entrées/Sorties ; pour celà, on utilisera en particulier les techniques de mémoire cache placée devant les disques et celle de filtrage si possible effectué en parallèle à la lecture des données sur disques. D'autre part, l'évaluation de requête permettra d'approcher la décomposition optimale, qui donne le moindre coût d'exécution de cette requête.

(5) Accélérer les recherches par un placement multi-attribut permettant de regrouper les données le plus souvent accédées ensemble et aussi par l'utilisation d'accélérateurs de types index secondaires permettant de retrouver rapidement les tuples ayant une même valeur d'attribut.

(6) Assurer l'intégrité des données en face des pannes de la machine ou des calculateurs hôtes, des accès concurrents aux données et des erreurs des opérateurs à l'entrée des données, et aussi permettre les vues multiples d'un même ensemble de données.

(7) Obtenir une dégradation douce de la machine sur pannes d'éléments non essentiels sans perte de fonctions par reconfiguration automatique ; une telle méthode nécessite des redondances du matériel bien choisies.

Il s'agit plus particulièrement d'atteindre au moins partiellement ces sept objectifs en faisant cohabiter des solutions cohérentes à chacun d'eux. Le système permet par exemple la gestion d'une base relationnelle composée des relations représentées fig. 1. Un exemple type de questions qui peut être soumis sur une telle base est : "Trouver le nom et la ville des buveurs auxquels on a expédié du Nuit-Saint-Georges 1976 en Octobre 81, ainsi que le nom des viticulteurs correspondants".

VINS (NVIN, CRU, ANNEE, NVIT)
VITICULTEURS (NOM, PRENOM, QTE.DISP, NVIT)
BUVEURS (NOM, PRENOM, VILLE, NBUV)
COMMANDES (DATE, NCOM, NVIN, QTE, NBUV)
EXPEDITIONS (DATE, NCOM, QTE)

Fig. 1 : Exemple de base

3. PRINCIPES DE BASE

Le système SABRE est construit à partir d'un ensemble de concepts développés au sein du projet. Nous présentons ci-dessous les principaux concepts retenus.

3.1. Protocole de manipulation de données

Tout d'abord, la machine est conçue pour être connectée à un réseau local ou général et obéit à un protocole de manipulation de données. Un tel protocole se situe au niveau 7 de l'architecture ISO {5} (cf. fig. 2) : il s'agit donc d'un protocole d'application permettant de manipuler des données à distance. Plus précisément, le protocole se compose d'un ensemble de commandes permettant la gestion de données structurées et le contrôle de transaction {6}. Dans le futur, il devrait être enrichi afin de permettre la gestion d'images et de textes stockés sous forme numérique (cf. fig. 3).

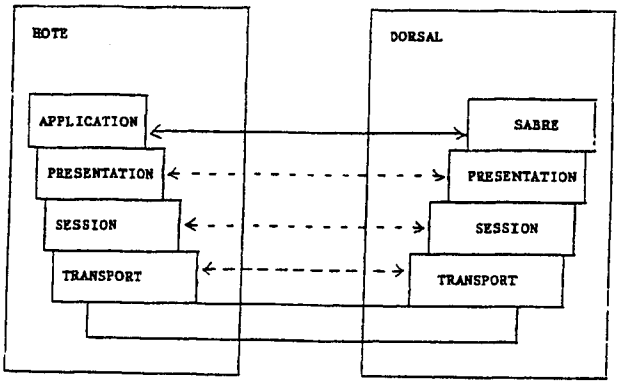


Fig. 2 : Les niveaux de protocole

Le contrôle des transactions est effectué à l'aide des quatre commandes suivantes :

- DEBUTER (TRANSACTION, MODE, UTILISATEUR, VUE) permet d'initialiser une nouvelle transaction pour l'utilisateur cité sur la vue indiquée dans le mode précisé.

- VALIDER (TRANSACTION) permet de savoir si toutes les commandes exécutées jusque-là par une transaction se sont bien passées ; de plus, cette commande prépare le commitment de la transaction en enregistrant les mises à jour effectuées en mémoire sûre, de sorte qu'elles pourront soit être appliquées à la base, soit annulées, quelles que soient les pannes qui surviendront, selon la décision du calculateur hôte.

- COMMETTRE (TRANSACTION) enregistre définitivement les mises-à-jour d'une transaction dans la base de données et signale la fin de la transaction qui aura dû être précédemment commise.

- ABANDONNER (TRANSACTION) provoque l'arrêt d'une transaction et l'annulation de ses mises-à-jour.

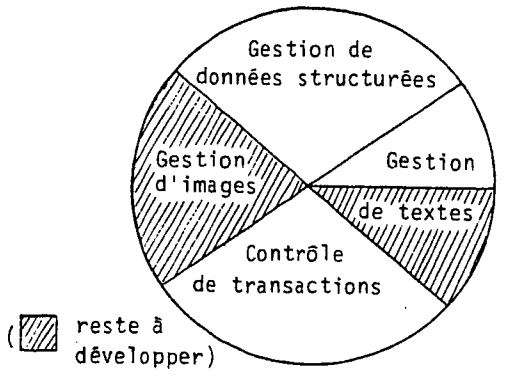


Fig. 3 : Types de commandes

Après définition d'une base relationnelle, le protocole permet d'exécuter à l'intérieur d'une transaction les commandes typiques suivantes (sous une forme simplifiée) :

- INSERER (RELATION, TUPLES) permet d'ajouter des tuples dans une relation existante.

- SUPPRIMER (RELATION, CRITERE) permet de supprimer dans une relation les tuples vérifiant le critère indiqué.

- MODIFIER (ATTRIBUTS = VALEURS ou EXPRESSION, RELATION, CRITERE) permet de modifier les attributs des tuples d'une relation vérifiant le critère indiqué en les forçant aux valeurs indiquées ou calculées.

- RECHERCHER (ATTRIBUTS ou EXPRESSIONS D'ATTRIBUTS, CRITERE) permet de rechercher dans une ou plusieurs relations les valeurs des attributs cités ou des expressions citées des tuples vérifiant le critère indiqué.

Il est également possible de créer et détruire des chemins d'accès spéciaux aux données, après les avoir définis, par deux commandes spécifiques :

. CREER_ACCELERATEUR,

. DETRUIRE_ACCELERATEUR

3.2. Hiérarchie de vues

La définition d'une base de données s'effectue par définition de hiérarchies de vues qui peuvent être reliées pour former des vues multi-bases. Une première vue racine peut décrire toutes les relations de la base : il s'agit alors du traditionnel schéma de la base. Des vues successives peuvent être dérivées de la racine ou d'une ou plusieurs vues elles-mêmes dérivées. Une vue "dérivée" ne peut être définie que par un administrateur des vues dont elle est dérivée (les vues "origines"). Des administrateurs pourront alors être désignés pour cette vue dérivée, laquelle pourra alors à son tour devenir vue origine.

Parmi les relations d'une vue, certaines peuvent être réellement implantées sur disques, d'autres seront simplement reconstituées au moment de leur utilisation. Le choix de l'implantation est laissé au définisseur de la vue, sa décision pouvant être dictée par des critères de performance (ex : fréquence d'utilisation de la vue).

Afin d'exprimer les règles de composition des relations d'une vue, un ensemble de questions portant sur la vue origine est associé à une vue dérivée. Par ailleurs, lorsque les modifications à travers une vue sont autorisées par le définisseur de cette vue, un ensemble de contraintes d'intégrité portant sur cette vue et un ensemble de requêtes de modifications à répercuter sur la vue origine seront définis. Ces définitions sont destinées à assurer la cohérence des données entre les vues. Pour l'instant, seules les mises-à-jour sur la vue racine (base) sont autorisées.

Ainsi, nous obtenons une définition très décentralisée et souple d'une base de données, avec la possibilité très utile d'associer à une vue des données privées au groupe d'utilisateurs y ayant accès.

3.3. Méta-base

Les bases de données sont décrites dans une méta-base. La métabase se compose pour l'instant de six relations essentielles :

- la relation VUE contient les caractéristiques des vues gérées

- la relation RELATION contient les caractéristiques des relations des vues gérées

- la relation ATTRIBUT contient les caractéristiques des attributs composant les relations des vues gérées

- la relation CLES permet de préciser quels sont les attributs des relations

- la relation DROITS contient les droits d'accès des usagers aux vues

- la relation REGLE contient les règles de décomposition des relations dérivées, les règles de répercussion des modifications sur la vue origine ainsi que les contraintes d'intégrité associées aux relations.

La méta-base peut être manipulée comme une base normale, car elle contient son propre schéma. La machine connaît à priori, néanmoins, ce schéma.

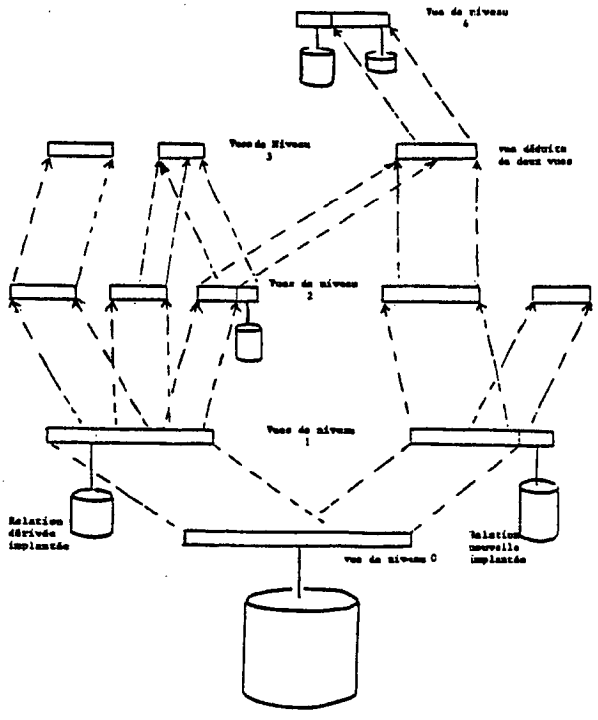


Fig. 4 : Une hiérarchie de vues (avec des liens)

3.4. Décomposition optimisée

Une requête utilisateur est décomposée en une suite d'opérateurs relationnels. Les opérateurs relationnels implantés dans la machine sont la sélection (restriction + projection sans élimination des doubles), la jointure, le tri, l'élimination des doubles, l'union, la différence, l'insertion, la suppression ainsi que des fonctions arithmétiques et agrégats.

Un algorithme d'optimisation permet de donner une décomposition optimale de la requête, qui minimise le coût d'exécution (ES + CPU) et le

coût d'utilisation des ressources communes. D'autre part, cet algorithme tend à maximiser le parallélisme.

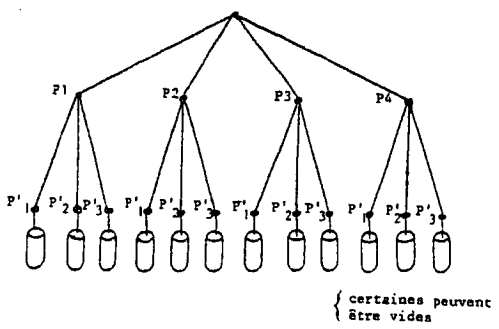
3.5. Arbres de prédicats

Afin de gérer les chemins d'accès, nous utilisons des arbres de prédicats. Un arbre de prédicat peut être vu comme une méthode hiérarchique pour diviser les relations en plus petites relations afin de réduire les espaces de recherche. Par exemple, une relation peut être partitionnée une première fois à l'aide d'un ensemble de prédicats disjoints P1, P2 ... Pn en n sous-relations ; chaque sous-relation Ri contiendra les tuples qui vérifient le prédicat Pi, et qui ne vérifient donc pas les prédicats P1 ... Pi-1 Pi+1 ... Pn. Chaque sous-relation peut à son tour être partitionnée à l'aide d'une liste de prédicats disjoints. Dans le but de simplifier la description des partitionnements successifs, une même liste de prédicats est appliquée à toutes les sous-relations. Ainsi, nous obtenons par partitionnement successifs un arbre de prédicats balancé dont chaque noeud est racine de sous-arbres identiques.

Lors d'une recherche la qualification de la recherche (c'est-à-dire le critère de sélection) est comparée chacun des prédicats de l'arbre. Seules les sous-relations qui correspondent à des prédicats non disjoints avec celui constituant la qualification doivent être balayées. Si l'arbre de prédicats est défini de sorte que chacune des sous-relations correspondent à une question fréquente, de nombreuses questions seront grandement accélérées.

A titre d'exemple, la fig. 5 représente un arbre de prédicats possible pour une relation VINS. La question "lister des vins de cru "Chablis" et de millésime 1982" nécessite de balayer seulement la sous-relation la plus à droite.

Par contre, la question "lister les vins de millésime 1970" nécessite de balayer toutes les sous-relations associées à la première branche des sous-arbres issus des noeuds étiquetés P1, P2, P3 et P4.



P1 : CRU = "Beaujolais-Village"
P2 : CRU = "Moulin-à-Vent"
P3 : CRU = "Bordeaux"
P4 : ¬P1 ∧ ¬P2 ∧ ¬P3 ∧ ¬P4 (CRU="Autre")
P'1 : MILLESIME ≤ 1970
P'2 : (MILLESIME > 1970) ∧ (MILLESIME < 1980)
P'3 : MILLESIME ≥ 1980

Fig. 5 : Exemple d'arbre de prédicats

3.6. Partition associative

Les tuples sont rangés à l'intérieur de partitions associatives dont la taille est en général une piste mais peut être moins. L'adressage à l'intérieur d'une partition s'effectue par le contenu. Un critère de recherche est composé de prédicats élémentaires du type (attribut.θ.valeur) où attribut désigne un numéro d'attribut dans le tuple, θ un opérateur de comparaison choisi parmi (=,<,≤,≠,>,≥) et valeur une valeur numérique ou alphanumérique ou textuelle.

A chaque unité de disques est associé un ensemble de processeurs de filtrage capables d'exécuter les sélections. Le nombre de processeurs de filtrage est égal au nombre de flots de données provenant en parallèle de l'unité de disques (en général 1). Un processeur de filtrage peut être un microprocesseur rapide effectuant les sélections par programme ou un processeur câblé spécialisé utilisant un automate de sélection {7,8,9}. Un processeur de filtrage possède une mémoire locale capable de contenir, en plus des données nécessaires à l'exécution de la sélection le plus possible en parallèle à la lecture d'une piste (à la volée), les résultats de la sélection (au plus, une partition). Une unité de disques avec deux processeurs de filtrage associés est représentée fig. 6.

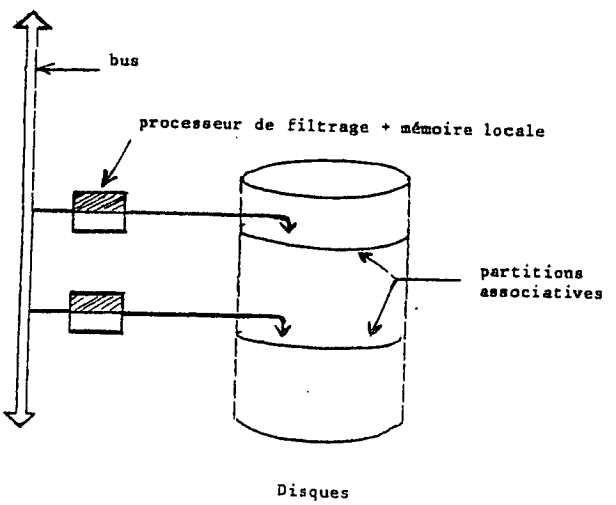


Fig. 6 : Processeurs de filtrage

4. ARCHITECTURE FONCTIONNELLE ET PARALLELISME

4.1. Principes de l'architecture fonctionnelle

L'architecture fonctionnelle de SABRE se compose d'un ensemble de couches emboîtées, allant des disques à l'utilisateur devant son terminal. Chaque couche est caractérisée par son interface externe. L'interface d'une couche i est appelable par toutes les couches à l'aide des requêtes constituant son interface externe. Une requête permet de déclencher une opération sur des objets. Les couches successives sont capables de gérer des objets de plus en plus petits.

4.2. Les couches de SABRE

Le dialogue avec les utilisateurs est assuré par une première couche externe. Dans le cas d'une utilisation de SABRE comme un serveur base de données, connecté par un réseau à un calculateur hôte, cette couche serait implantée sur le calculateur hôte. C'est elle qui assure le dialogue avec l'utilisateur, l'analyse syntaxique et sémantique des requêtes et leur codage en messages du protocole de manipulation de données. Dans une version commerciale, l'interface avec des langages de programmation de type PASCAL ou COBOL devrait être assurée à ce niveau.

La couche la plus externe du serveur SABRE gère les vues. Une vue est un ensemble de relations, éventuellement déduites par questions de relations existantes sur les disques, associées à un groupe d'usagers qui peut accéder aux relations de la vue. Les requêtes de cette couche référencent donc les objets d'une ou plusieurs vues. La couche a essentiellement pour rôle le contrôle des droits d'accès des usagers, la modification des questions pour passer des relations déduites aux relations réelles, et les contrôles de l'intégrité des données lors des mises-à-jour.

La couche suivante gère les bases. Une base peut être définie comme un ensemble de relations existant sur les disques. Les requêtes à la couche base sont donc des recherches ou des mises-à-jour dans une base, qualifiée par un critère complexe, mettant en jeu plusieurs relations. C'est cette couche qui décompose chaque manipulation complexe portant sur plusieurs relations en opérations de l'algèbre relationnelle étendue.

La couche suivante gère les relations. Elle est capable d'exécuter les opérations unaires sur relations (sélection, tris, dédoublage, aggrégats ...), mais aussi les opérations binaires (jointures). Elle détermine les partitions logiques à accéder pour exécuter une requête sur une relation et est donc plus particulièrement chargée de la gestion des chemins d'accès à l'aide des arbres de prédicats.

La dernière couche gère les partitions. Elle est essentiellement capable d'exécuter les sélections et les mises-à-jour sur des ensembles de partitions adressées par une adresse logique appelée signature. C'est dans cette couche que sont exécutées les opérations de filtrage, de mémorisation, mais aussi de contrôle de concurrence et d'atomicité des transactions.

L'architecture fonctionnelle d'ensemble de SABRE est résumée fig. 7. Les cinq couches et leurs interfaces sont illustrées. Rappelons que nous autorisons une couche à appeler toutes les couches plus internes.

4.3. Possibilités de parallélisme

L'exécution d'une requête typique met en jeu les diverses couches comme illustré fig. 8.

Afin d'augmenter le débit global de la machine, deux types de parallélisme sont possibles :

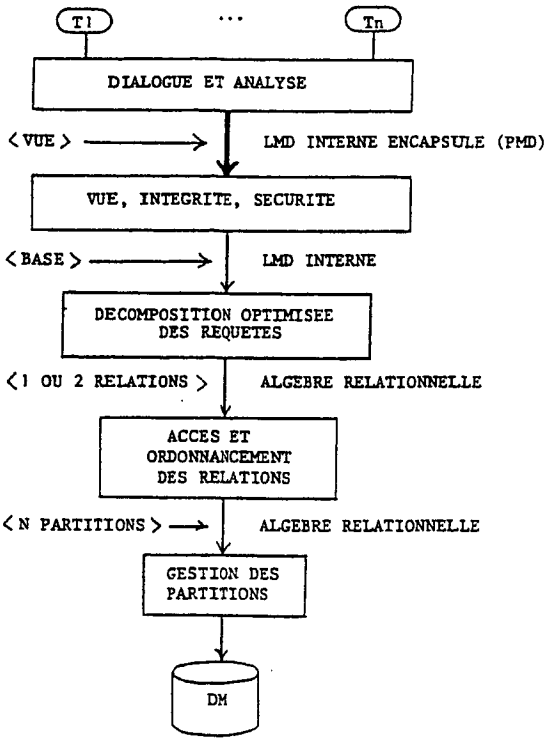


Fig. 7 : Architecture fonctionnelle. Vue d'ensemble

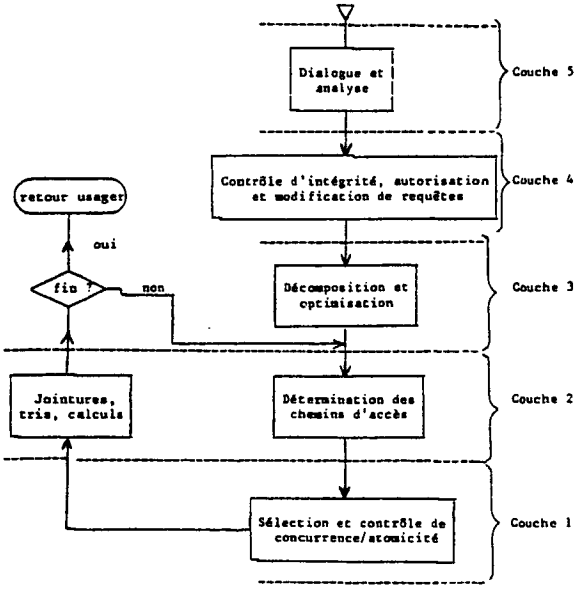


Fig. 8 : Exécution d'une requête type

(1) Le parallélisme inter-requête consiste à traiter plusieurs requêtes en parallèle. Il peut être accompli en associant un processeur matériel à chaque couche. Ainsi, SABRE peut être décomposé en processeurs indépendants accomplissant chacun les fonctions spécifiques d'une couche. De plus, un noyau d'exécution réparti {10} doit bien sûr coordonner les différents processeurs en assurant les communications et synchronisations.

(2) Le parallélisme intra-requête consiste à traiter une même requête en parallèle, à l'aide de plusieurs processeurs. Un tel parallélisme intra-requête peut être obtenu de deux manières :

a) par décomposition d'une requête en étapes parallélisables
b) par mise en collaboration d'un ensemble de processeurs afin d'accomplir une même étape correspondant à une couche.

Le cas (a) est obtenu par décomposition optimisée d'une requête en arbre d'opérations relationnelles. Par exemple, la requête "Trouver le nom et la ville des buveurs auxquels on a expédié du Nuit-Saint-Georges 1976 en Octobre 81" se décompose dans l'arbre relationnel représenté fig. 9. Certains des opérateurs SABRE représentés par des rectangles peuvent être accomplis en parallèle, par exemple, les sélections initiales.

Le cas (b) est obtenus par élaboration d'algorithmes parallèles pour exécuter les fonctions d'une couche. Deux méthodes sont encore possibles.

La division des données à traiter en parties (non forcément indépendantes) et la mise en jeu de plusieurs processeurs pour traiter chaque partie est une première méthode possible. Ce type de parallélisme par division peut être appliqué afin de paralléliser les opérateurs de jointure et de sélection. Il est en effet possible de faire exécuter une jointure par un ensemble de processeurs travaillant en parallèle et coopérant. De même, le partitionnement des relations et le placement sur des unités de mémoires secondaires accessibles en parallèle rend possible l'exécution d'une sélection par plusieurs processeurs parallèles.

La division de la couche en fonctions exécutables en parallèle est une deuxième méthode possible. Ainsi par exemple, les fonctions de contrôle de concurrence et de gestion mémoire peuvent être séparées à l'intérieur de la couche la plus interne de gestion des partitions ; ces fonctions peuvent alors être traitées par deux processeurs fonctionnant plus ou moins en parallèle.

L'isolement de la plupart des unités parallélisables nous a conduit à développer une architecture opérationnelle idéale pour SABRE. Cette architecture opérationnelle idéale est composée de classes de processeurs identiques. Une couche fonctionnelle est associée à une ou plusieurs classes de processeurs.

Nous allons dans la suite présenter cette architecture fonctionnelle idéale ainsi que les architectures opérationnelles plus réelles des systèmes en cours de réalisation, qui tendent à s'en approcher par des générations successives.

5. ARCHITECTURES OPERATIONNELLES

5.1. Principes

Du point de vue architecture opérationnelle, il faut distinguer l'architecture opérationnelle idéale du système SABRE des architectures opérationnelles des diverses générations qui peuvent être construites. L'architecture opérationnelle idéale est composée d'un ensemble de processeurs spécialisés organisés en classes successives de processeurs parallèles. Chaque architecture opérationnelle est composée d'un ensemble de processeurs logiciels supportés par un ou des processeurs matériels communiquant par des bus. Une architecture opérationnelle correspond à une génération du système SABRE. A l'intérieur de chaque génération, il existe des versions de logiciel.

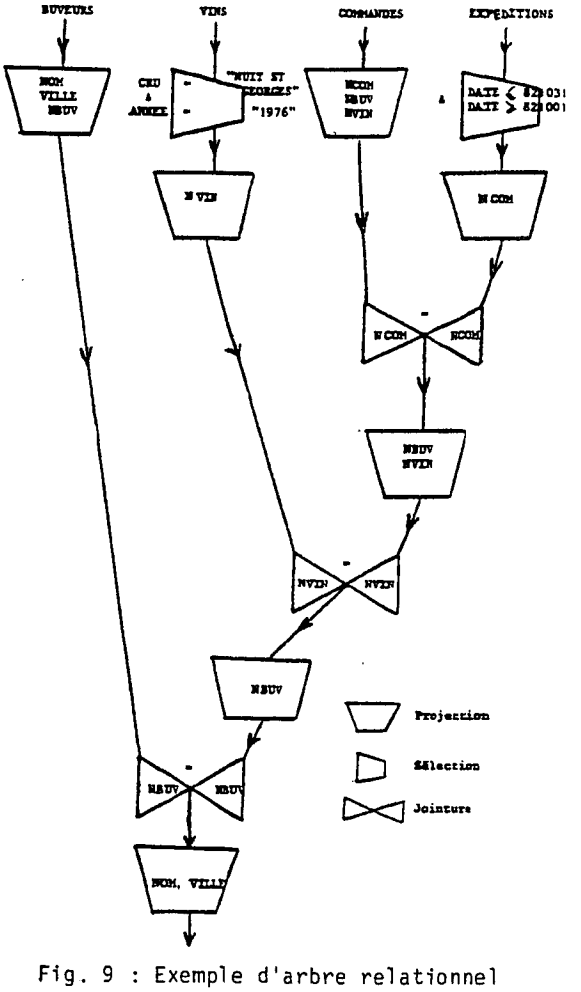


Fig. 9 : Exemple d'arbre relationnel

5.2. Architecture opérationnelle idéale

Comme indiqué ci-dessus, l'architecture opérationnelle idéale se compose d'un ensemble de processeurs que nous allons succinctement décrire. La présentation est organisée du haut vers le bas, c'est-à-dire de l'interface externe vers les mémoires secondaires. L'ensemble des processeurs composant l'architecture fonctionnelle est représenté fig. 10.

Les processeurs hôtes (HT) s'interfacent avec l'usager. Ils sont de deux types HT0 et HT1. HT0 réalise une interface conversationnelle guidée par questions/réponses. Il saisit et analyse syntaxiquement un langage de manipulation de données proche du langage QUEL. Il génère des commandes codées selon le format du protocole de manipulation de données. HT1 réalise un langage interactif proche de SQL. Il s'agit en fait d'un analyseur paramétré qui traduit les verbes de manipulation de données selon le

format du protocole de manipulation de données. Le langage est très souple puisque l'analyseur peut aller jusqu'à générer les jointures manquantes.

Les processeurs d'intégrité et de vues (PIV) sont au niveau le plus externe et travaillent sur les vues. Un tel processeur reçoit les requêtes des ou du processeur(s) hôte(s) par l'intermédiaire du réseau ou du logiciel le simulant. Ces requêtes, exprimées en termes de commandes et paramètres du protocole de manipulation de données, portent sur une vue d'une base de données. Après contrôle des droits d'accès des transactions, le rôle du PIV est de traduire la requête portant sur des relations virtuelles décrites dans la vue en requêtes portant sur des relations réellement implantées dans la base. Il est aussi de contrôler l'intégrité des données lors des mises-à-jour et d'assurer la cohérence des diverses relations implantées au niveau de vues différentes.

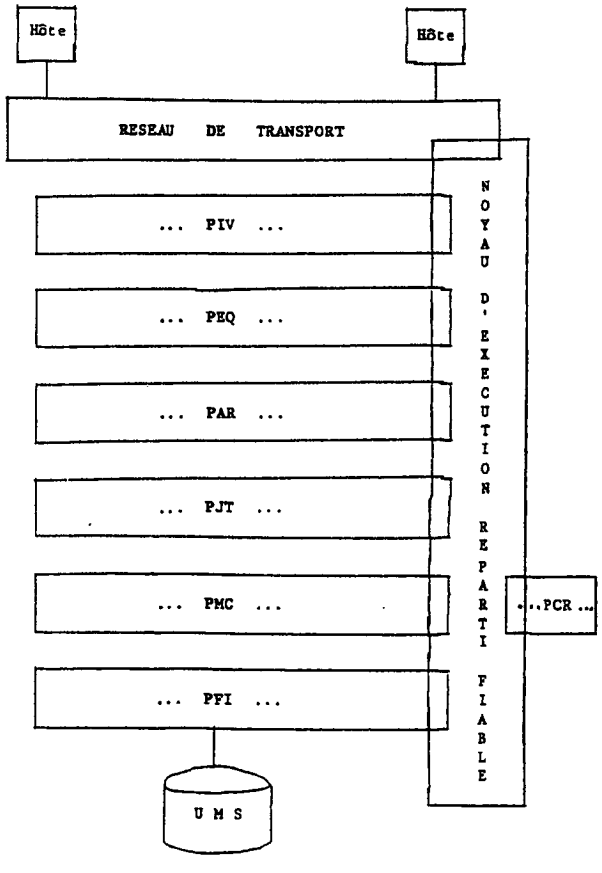


Fig. 10 : Architecture opérationnelle idéale de SABRE

Les processeurs d'évaluation de questions (PEQ) travaillent au niveau du schéma de la base implantée. Un tel processeur est chargé de décomposer une question en arbres d'opérations relationnelles étendues (sélection, jointure, union, différence, tri, ...) et aussi de décomposer une mise-à-jour en un remplacement de valeurs suivi d'une insertion. Ce processeur évalue les décompositions possibles d'une même requête et choisit la décomposition optimale.

Les processeurs d'accès aux relations (PAR) travaillent au niveau d'une relation (opérations unaires) ou de deux relations (opérations binaires). Un tel processeur gère les arbres de prédicats associés à une relation. Lors d'une sélection, il détermine les adresses logiques (signatures) des partitions susceptibles de contenir des tuples vérifiant le critère de sélection. Lors d'une jointure de deux relations, il détermine les couples de partitions susceptibles de contenir des tuples participant à la jointure. Lors d'une mise-à-jour, il répercute cette opération conformément aux prédicats de la relation qu'il gère.

Les processeurs de jointure, tri et calcul d'agrégats (PJT) travaillent sur des listes de partitions à l'intérieur d'une relation. Un tel processeur accomplit le choix de l'algorithme optimal de jointure {11} ou semi-jointure {12}, participe à l'exécution des jointures, des tris, éliminations des doubles ainsi que l'exécution des fonctions arithmétiques et agrégations. Une agrégation divise une relation en sous-ensembles disjoints suivant les valeurs d'attributs spécifiés, et pour chaque sous-ensemble, calcule une fonction arithmétique (SOMME, MOYENNE, MINIMUM, MAXIMUM, COMPTE) associée à un attribut.

Les processeurs mémoire cache (PMC) sont chargés de la gestion des mémoires caches et de l'allocation de l'espace mémoire secondaire. Un tel processeur alloue des pages aux relations de la base ou aux relations de travail en mémoire. Il effectue aussi le placement des pages dans les partitions associatives sur disques lors des écritures ou lorsque la mémoire cache est saturée. Il choisit également l'emplacement physique des nouvelles partitions créées. Lorsqu'une sélection est effectuée sur une page résidente en mémoire, celle-ci est exécutée par le processeur mémoire cache. Celui-ci effectue également diverses fonctions de services : hachage d'une relation, extraction de tuples ...

Les processeurs de contrôle et reprise (PCR) effectuent les contrôles de concurrences à l'aide d'un algorithme basé sur un double ordonnancement des transactions {13} et exécutent la journalisation des mises-à-jour de relations. Pour cela, une relation image est associée à chaque relation implantée de la base et contient l'historique des modifications apportées à cette relation depuis la dernière sauvegarde de la base. Ce processeur gère également la validation, le commitment et l'abandon des transactions.

Les processeurs de filtrage (PFI) travaillent au niveau de la partition associative. Un tel processeur effectue les sélections des attributs relevant des tuples vérifiant le critères de sélection à la volée durant la lecture d'une piste ou en mémoire après cette lecture. Un processeur de filtrage est théoriquement associé à chaque flot de données en provenance d'une unité de disques.

Au niveau de l'architecture opérationnelle idéale, le nombre des processeurs virtuels n'est pas fixé : seules leur position, leurs interfaces et leurs fonctions sont définies. De même le noyau d'exécution réparti n'est pas spécifié.

Il s'agit d'un système de contrôle capable d'assurer les communications entre les différents processeurs.

5.3. Architecture opérationnelle de la première génération

Une génération de SABRE correspond à un système réel implanté sur un ou plusieurs processeurs réels. La génération 1 du système SABRE est caractérisée par l'existence d'un seul processeur réel. Deux versions ont été ou sont développées sur MULTICS, la version 0 qui a servi pour valider les interfaces, et la version 1 qui est un SGBD relationnel complet.

L'architecture opérationnelle de SABRE.G1 est représentée fig. 11. SABRE.G1 est actuellement opérationnelle sur MULTICS dans une version 1. SABRE.G1 a été essentiellement réalisée dans le but de valider les interfaces et algorithmes de l'architecture fonctionnelle et de mettre au point une version non parallèle des processeurs essentiels. Il s'agit donc d'une version multi-usager mono-processeur réel. Les différents processeurs sont écrits en PASCAL et donc en théorie au moins facilement portables.

La terminaison de SABRE.G1 est prévue pour fin 1983 sur MULTICS. Il s'agira d'un système de bases de données relationnelles complet comportant la plupart des notions originales présentées ci-dessus.

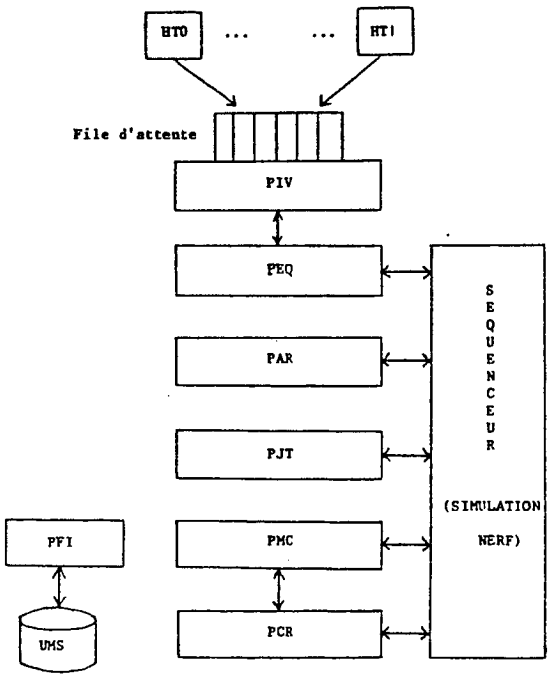


Fig. 11 : Architecture opérationnelle de SABRE.G1

5.4. Architecture opérationnelle de la deuxième génération

L'architecture opérationnelle de la deuxième génération du système appelée SABRE.G2 est représentée fig. 12. SABRE.G2 est une véritable machine bases de données multi-microprocesseur. Elle devrait comporter trois processeurs et être utilisable simultanément par au moins deux usagers dans une première maquette. Les logiciels correspondant aux processeurs virtuels devraient être les mêmes que ceux de SABRE.G1 (transport des programmes écrits en PASCAL). Le multi-microprocesseur retenu est la SM 90. L'architecture multi-microprocesseur originale de cette machine devrait permettre d'aborder une première étude réelle des performances d'un tel système.

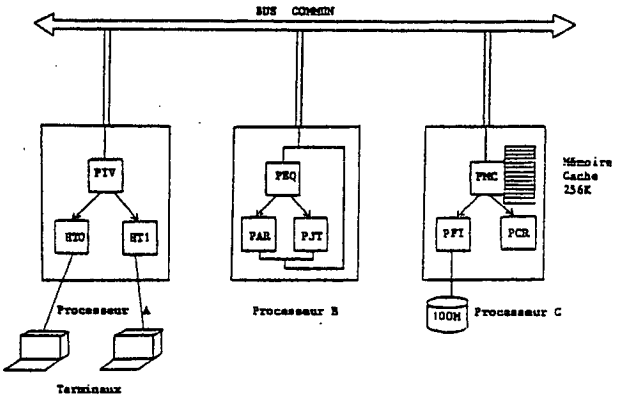


Fig. 12 : Architecture opérationnelle de SABRE.G2

6. CONCLUSION

Cette communication a présenté le projet SABRE. Le premier objectif du projet est d'obtenir des gains appréciables en performances et fonctionnalités pour la gestion de grandes bases de données relationnelles. Les principaux concepts de base ont été définis : utilisation du parallélisme, filtrage des données à la volée, nouvelle technique pour l'accès rapide aux données, anté-mémoire ... Un autre objectif important est la portabilité du système SABRE, qui est écrit en PASCAL.

La distinction entre architectures fonctionnelle et opérationnelle a permis l'indépendance entre fonctions à réaliser et logiciel pour les réaliser. Ainsi, les concepts relatifs à l'optimisation des fonctions ont pu être dégagés des couches de logiciels supportant ces fonctions qui constituent l'architecture fonctionnelle. L'architecture fonctionnelle est traduite en architectures opérationnelles, en affectant des processeurs logiciels et matériels aux couches. En conséquence, le système SABRE devrait être facilement portable sur différents systèmes réels (HB 68, multi-microprocesseurs, ...).

La réalisation du système est faite par étapes correspondant à des générations opérationnelles successives. Cette approche progressive a commencé par la mise au point d'une version 0 de la première génération, mono-usager sur MULTICS, actuellement opérationnelle. Cette version a permis la validation des interfaces et algorithmes dans un contexte mono-processeurs. La version 1 de la première génération est un SGBD relationnel complet comportant des approches originales. La deuxième génération devrait être une véritable machine bases de données, implan-

tée sur un système multi-microprocesseur avec parallélisme vrai. Une version future, plus sophistiquée, devrait permettre l'intégration des données textes et images, pour constituer un système de gestion de bases de données intégrées de troisième génération.

7. REFERENCES

{1} D.D. Chamberlin, A.M. Gilbert, R.A. Yost : "A history of System-R and SQL/Data System", 7th Int. Conf. on Very Large Data Bases, Cannes, September 1981.

{2} M. Stonebracker, E. Wong, P. Kreps : "The design and implementation of INGRES", ACM Transactions on Data Base Systems, vol. 1, n° 3, September 1976.

{3} J. Le Bihan, C. Esculier, G. Le Lann, W. Litwin, G. Gardarin, S. Sédillot, L. Treille : "SIRIUS : a french nationwide project on distributed data bases", 6th Int. Conf. on Very Large Data Bases, Montreal, October 1980.

{4} G. Gardarin : "An introduction to SABRE : a multi-microprocessor database machine", 6th Workshop on Computer Architecture for Non Numeric Processing, Hyères, France, June 1981.

{5} ISO/TC97/SC16 N227 : "Reference model of open system interconnection", June 1979.

{6} N. Temmerman : "Un protocole de manipulation de données pour la machine bases de données SABRE", Rapport de DEA, Institut de Programmation, Paris VI, Septembre 1981.

{7} F. Bancilhon, M. Scholl : "Design of a back-end processor for a database machine", Proc. of the ACM-SIGMOD, Santa Monica, California, May 1980.

{8} P. Faudemay : "Sur une nouvelle classe de filtres multi-expressions", Journées Machines Bases de Données, Sophia-Antipolis, Septembre 1980.

{9} J. Rohmer : "Présentation du processeur de filtrage SCHUSS", Journées Bases de Données, Toulouse, Septembre 1982.

{10} P. Bernadat : "La machine bases de données SABRE et son noyau d'exécution répartie (NER)", Rapport de DEA, Institut de Programmation, Univ. de Paris VI, Septembre 1981.

{11} P. Valduriez, G. Gardarin : "Multiprocessor join algorithms of relations", 2nd Int. Conf. on Data Bases, Improving Reliability and Responsivness, Jerusalem, June 1982.

{12} P. Valduriez : "Semi-join algorithms for multiprocessor systems", Int. Conf. on Management of Data, Orlando, June 1982.

{13} Y. Viémont, G. Gardarin : "A distributed concurrency control based on transaction commit ordering", Proc. of FTCS'12, Santa Monica, June 1982.

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique